Deadline Constrained Video Analysis via In-Transit Computational Environments

Ali Reza Zamani, Mengsong Zou, Javier Diaz-Montes, Ioan Petri, Omer Rana, Ashiq Anjum, and Manish Parashar

Abstract—Combining edge processing (at data capture site) with analysis carried out while data is enroute from the capture site to a data center offers a variety of different processing models. Such in-transit nodes include network data centers that have generally been used to support content distribution (providing support for data multicast and caching), but have recently started to offer user-defined programmability, through Software Defined Networks (SDN) capability, e.g. OpenFlow and Network Function Visualization (NFV). We demonstrate how this multi-site computational capability can be aggregated to support video analytics, with Quality of Service and cost constraints (e.g. latency-bound analysis). The use of SDN technology enables separation of the data path from the control path, enabling in-network processing capabilities to be supported as data is migrated across the network. We propose to leverage SDN capability to gain control over the data transport service with the purpose of dynamically establishing data routes such that we can opportunistically exploit the latent computational capabilities located along the network path. Using a number of scenarios, we demonstrate the benefits and limitations of this approach for video analysis, comparing this with the baseline scenario of undertaking all such analysis at a data center located at the core of the infrastructure.

Index Terms—edge computing, in-transit, software-defined networks, video analysis, cloud federation, CometCloud

1 Introduction

With the maturity of the Internet of Things (IoT) paradigm and associated devices, data sensing can now be combined with data processing/analysis on the same device. As IoT devices increase in function and capability, existing infrastructures such as monitoring/ storage and network capabilities can be combined to create a more general purpose data analysis and computational environment. Such a perspective assumes that IoT devices and in-transit network nodes, over which such data is channeled, can be used to support data processing along with the data centres to which this data is sent, typically located at the core of the infrastructure. This comes with the recent interest in moving away from centralized, large-scale data centers to a more distributed multi-cloud setting (as demonstrated by significant interest in cloud federation and interoperability efforts). Such a multi-cloud environment is often formed by a network of smaller virtualized infrastructure runtime nodes, often with an ad hoc and unstructured architecture.

Combining IoT and Cloud computing capability enables the creation of smart environments that can respond to real-time events, by (a) combining services offered by multiple stakeholders (i.e. those that are at the network edge with services provided within a data centre) and, (b) providing scale to support a large number of users in a reliable and decentralized manner. They need to be able to operate in both wired and wireless network environments and deal with constraints such as access devices or data sources with limited power and unreliable connectivity. The Cloud application platforms need to be enhanced to support (a) the rapid deployment of services by providing domain specific programming tools and environments and (b) seamless execution of applications harnessing capabilities of multiple dynamic, and heterogeneous, resources to meet quality of service requirements of different users.

Additionally, network operators are increasingly becoming potential providers of general purpose computation infrastructure. They are minimizing the amount of network-specialized hardware hosted in their data centers and moving towards the use of commodity hardware. This strategy is supported in recent efforts in Software Defined Networking (SDN) and Network Functions Virtualization (NFV). SDN, in particular, is an approach devised to simplify network management through abstraction of lower-level functionality. Specifically, SDN separates control plane (where to send data) from data plane (data forwarding functions). This enables the software-based control plane to be run on commodity servers and to leverage the latest-generation of processors, which are faster than embedded-class processors in most switches [9]. On the other hand, NFV goes a step further and extends the as-a-service cloud model to offer networking functions on-demand using virtualization techniques. This approach promises, as the cloud, a reduction in capital expenses and a rapid deployment and delivery of new functionality [18].

Data centers managed and operated by network providers form a significant part of the current Internet infrastructure, as there is a large number of such data centers that are almost ubiquitous across the world. These data centers may not be as powerful as computational data centers, hosted by cloud providers or traditional high performance computing (HPC) providers. However, their ubiquity and the fact that we have to necessarily use them, when moving data over the Internet, makes them a useful source of pervasive computing at the edge of the network. Understanding how the availability of commodity servers within such "network data centers" can contribute towards data processing would enable an effective way to extend the boundaries of a cloud system – from a

- *A.R. Zamani, M. Zou, J. Diaz-Montes, and M. Parashar are with Rutgers Discovery Informatics Institute, Rutgers University, NJ USA.*
- *I. Petri and O. Rana are with School of Computer Science & Informatics, Cardiff University, UK.*
- *Ashiq Anjum is with the Computing & Mathematics Dept., University of Derby, UK*

high end, often localized data center, to multiple distributed data centers that can process data while it is in transit from source to destination. This also provides the possibility of additional revenue models for network providers – who are able to convert underutilized network resources to offer in-transit computation.

In this paper we propose a model to leverage the combined use of computational capabilities available at the network edge and within network data centers to support data transformation and analysis from source to destination. We demonstrate how this can lead to more efficient use of computational resources and extend the capability and capacity of the overall infrastructure. The contributions of this paper are:

- An in-network computational model to leverage computational resources located at the edge, within the network and those at a *traditional* cloud data centre.
- An optimization strategy that allows us to prioritize data processing based on the expected value of the data to user. We describe how this subjective notion can influence the location of where data processing takes place.
- An experimental and analytical validation of the proposed model using a video analysis use case.

The rest of the paper is organized as follows. Section 2 presents our motivating use case. Section 3 presents our federation model. Section 4 formalizes our problem, and Section 5 proposes an scheduling optimization strategy. Section 6 describes the deployment used for our experiments. Section 7 presents our evaluation and results. Section 8 collects the related work. Section 9 discuss the results obtained. Section 10 presents the conclusions and ongoing activities.

2 Video Analytics Use Case

We describe use cases centered on processing video sequences submitted from a single/ multiple camera(s). These video sequences can be encoded using different formats, and need to be processed within a deadline. We therefore consider a semi-real time video sequence analysis – compared to "batch" analysis, where a video sequence is first archived and subsequently analysed in an off-line manner. Two aspects of video stream processing (consisting of a sequence of image frames submitted from a camera) can be considered: (i) the stream is viewed and stored for archiving – generally requiring the captured data to be transmitted over a network infrastructure for viewing/archiving purposes; (ii) the stream is processed (and annotated) using pre-defined filters (to support object detection & colour-based classification, template matching, etc). Operations associated with (i) are often seen as a precursor to those for (ii). Video analytics also involves a user identifying features/ events of interest to be considered in the video sequence – such as detecting objects of interest, size/ colour-based classification, potential area of interest (spatially), and an estimated duration associated with such events. Figure 1 presents the high level stream processing workflow. Video sequences can be encoded using a variety of different formats (Full HD, QCIF, CIF/4CIF/D1, H.264, etc). Each encoding (generally at 25 frames/sec) leads to different storage requirements and number of pixels per frame. The level of automation involved in analysing the video stream can also vary. From full automation, where a user defines an "analysis request" and does not require further interaction with the system, to an interactive request, where a user is able to see partial results at each stage of analysis and able to interact and modify analysis parameters. In sections 2.1 and 2.2 we describe how (partial) video analysis can be carried out at the capture site and at in-transit nodes (located between the capture source and a Cloud-based data centre). The base line scenario (for comparison) is that all data is migrated from capture source to the datacenter for analysis. Figure 2 illustrates the baseline scenario.

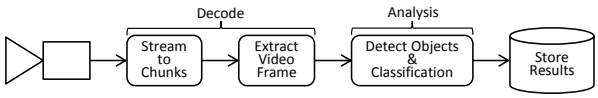


Fig. 1: High Level Stream Processing Workflow

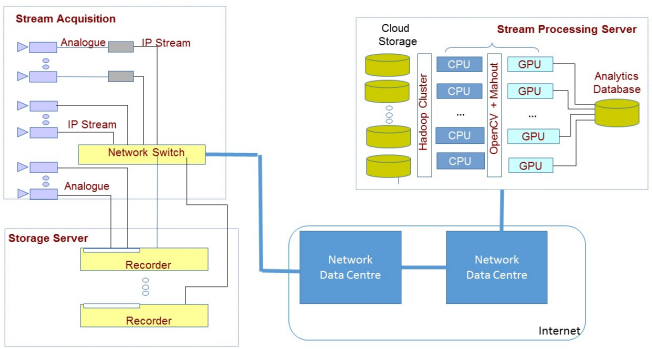


Fig. 2: Video Analytics Scenario – adapted from [2]

2.1 Scenario 1: Processing at Capture Node (Edge)

In this scenario, all data captured at the source is pre-processed at the source, prior to transmission across a network. Co-located hardware enables the captured data to be analysed before sending the processed results to the data centre. Processing at capture site can include: (i) data compression (at the camera) & buffering; (ii) data sampling – also as a means to support data size reduction; (iii) tagging of video frames prior to transmission. It may also be useful to combine data feeds from multiple cameras, to support correlation across multiple capture sites. Such aggregation and analysis would be useful at the first hop network component from the capture site.

2.2 Scenario 2: Processing at In-transit Nodes

In this scenario, data captured at the source is channeled through a number of intermediate "network data centres", prior to arrival at a video processing data centre. Each intermediate network data centre processes the data enroute – depending on the computational capability available. Subsequently, the data is archived and processed at the data centre, expected to have a much greater computational capability. The capability made available at the network data centre can vary over time, influenced by the other data streams that are being channeled through it at any time.

To make more effective use of the entire computational infrastructure, we propose that rather than sending all unprocessed data to a centralized location for processing, it is more efficient to initiate data processing at the edge of the infrastructure. Such an infrastructure includes data capture and generation devices, and the network path to its destination – e.g. using IoT gateways or devices, SDN switches, network data center, and clouds. In this way, we can incrementally augment the relevant information contained in the data, potentially reducing its size, while the data is being moved from source to destination. Additionally, this approach enables computational resource sharing, which not only improves resource utilization and throughput, but also increases the resilience to failures. Such an approach can also reduce latency and processing times resulting from unpredictable data (generation) sizes.

3 Resource Federation Model for Video Analytics

We extend our federation model [19] to expose in-transit and edge capabilities to participant sites of the federation. Figure 3 shows our architecture. We include a service, called Controller, that is aware of the network topology, using SDN technology, and it also has information about the available computational capabilities of each network data center. Each data center has an SDN router that is managed by the Controller and a set of resources to process tasks. The Controller can be consulted by the sites to optimize workload scheduling using the strategies proposed in Section 5.

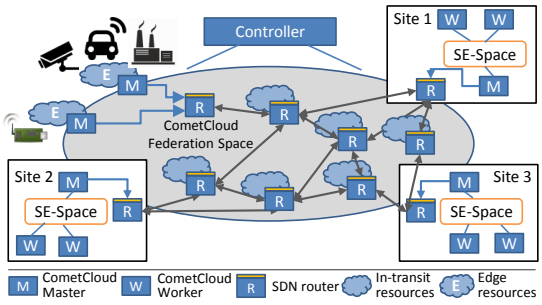


Fig. 3: Federated architecture that exposes edge and in-transit resources.

This federation model is built using the CometCloud framework [5]. CometCloud is an autonomic framework for enabling real-world applications on software-defined federated cyberinfrastructure, including hybrid infrastructures integrating public and private Clouds, data-centers and Grids. The CometCloud federation is created dynamically and collaboratively, where resources/sites can join or leave at any point, identify themselves (using security mechanisms such as public/private keys), negotiate terms of federation, discover available resources, and advertise their own resources and capabilities [6].

Our federation model is coordinated using CometSpaces [13] at two levels. CometSpaces provide a tuple-space like abstraction for coordination and messaging in the federation model – internally it implements a publish/subscribe messaging layer and an information lookup system built on a content-based distributed hash-table (DHT) based on a structured peer-to-peer overlay. First, a single management space (*CometCloud Federation Space*) spans across all resource sites creating and orchestrating the federation. This space is used to exchange any operational messages for discovering resources, announcing changes at a site, or routing users' request to the appropriate site(s). Second, multiple shared execution spaces (*SE-Space*) are created on-demand during application workflow executions to satisfy computational or data needs. Execution spaces can be created within a single resource site, or can burst to others, such as public clouds or external HPC systems.

Computational resources of our federation support at least a CometCloud Master, which acts as an agent or broker between local resources and the rest of the federation. It is also responsible for accepting computational requests from users and edge devices that want to access the federation. CometCloud Masters interact with the rest of the federation though the federation management space in a publish/subscribe fashion. Each CometCloud Master publishes information about the status of its resources, the services they offer, or computational needs of its users. Additionally, a CometCloud Master creates subscriptions to be notified when there is some event of interest, such as a request for computation. A CometCloud Master evaluates each request and decides if it can process it within the given QoS requirements, in which case it temporarily reserves the resources and answers the request with various details defining the Service Level Agreement (SLA), such as completion time and cost. If the client agrees to the SLA, then the computation proceeds, otherwise resource reservation is eliminated. In order to process a request, a CometCloud Master might create a SE-Space where it inserts the tasks and deploy CometCloud Workers to actually compute the tasks. In Edge and In-transit resources there are memory limitations, hence the SE-Space is not deployed and tasks are consumed as a stream by the CometCloud Workers.

As illustrated in figure 3, data captured at an edge resource (labelled `E`) is either directly submitted to a first hop gateway/router `R`, or pre-processed prior to transmission. Each `E` supports a CometCloud Master `M`. The first hop router can also aggregate data streams from multiple edge resources. This data is subsequently forwarded across a chain of in-transit resources (labelled as `R`) to a data center (labelled `Site i`). Each in-transit resource, similar to a data centre or edge device, must support a CometCloud Master – but with varying resource capability. At the data center, the Master also communicates with a number of Workers `W`. In this way, our resource federation can be logically seen as a collection of CometCloud Master nodes, which interact with each other to achieve the optimisation objectives outlined in section 4.

4 Problem Formulation

Let us consider a set of surveillance cameras $C : \{c_1, ..., c_n\}$, each of which generates a video stream that needs to be processed in a timely manner. A video stream can be partitioned into a sequence of m chunks, where each chunk contains a number of image frames. Processing each of these chunks is considered a computational job in our system – hence a given camera c_u is going to generate a set of jobs $J_u : \{J_{u1}, ..., J_{um}\}$. These jobs are introduced in the system periodically, as a sequence, i.e. if we consider 12 second chunks, then every 12 seconds a new job enters the system. Any given job J_{ux} is processed by a sequence of stages $\{s_1, ..., s_z\}$, forming a workflow as described in Section 2, Figure 1. Each stage is

composed of t tasks $\{j_1, ..., j_t\}$. The number of tasks depend on the size of the video chunks (e.g. in the classification stage we can have a task per frame and there are 25 frames in a second of video). The location of a camera is defined as source s. At the camera controller or aggregator, we need to decide which part is computed locally, in-transit, and/ or at the cloud. We consider that clouds are located at multiple network hops from the data source, at the core of the infrastructure. We consider that the video chunk and/or processed results need to be sent to a specific data center for storage and potentially additional offline processing with older data. This data center is defined as destination d in our system. The service level agreement (SLA) of a job J_{ux} includes: a deadline ($Deadline(J_{ux})$) by which results have to be placed at the destination – this is typically determined by the size of the video chunk; and a budget ($Budget(J_{ux})$) that sets the maximum amount available to the user to spend on computing job J_{ux}.

Central to our approach is the concept of *value* associated with the processing of data – a subjective criterion identified by a user. We define the value of data (i.e. a video chunk in our example application) as the significance a user associates with processing of particular data items within a deadline (captured as a subjective probability), in preference to other data items. For instance, in a surveillance scenario, the value associated with processing a video sequence (to detect/classify objects) would be higher if there was a public event in progress. Therefore, in our approach we pay attention to the *value* parameter to prioritize the processing of video streams by, for example, allocating high *value* workload closer to the data source and allocating low *value* workload to cheaper resources or rejecting low *value* workload when there are insufficient resources available.

We define three types of computational resources forming our federated infrastructure, namely edge devices (local to the data capture site), network data centers (in-transit resources), and computational data centers (cloud resource providers or sites). Formally, we define these resources as a set R with q resources $\{r_1, ..., rq\}$. We assume that SDN components are present in our infrastructure to ensure dynamic control over the network and provide QoS guarantees. The following symbols are used to characterise the problem:

- $P(r_i)$ is the average number of tasks that resource r_i completes per unit of time.
- $E(J_{ux}, r_i)$ is the time job J_{ux} spent computing at resource r_i.
- $BaseCost(r_i)$ is the cost per unit of time for using resource r_i for computation.
- $T(r_i, r_k)$ is the time spent transferring data between resources r_i and r_k.
- $BaseCostNet(r_i, r_k)$ is the cost of reserving a network channel per unit of time, between resources r_i and r_k.
- $Value(J_{ux})$ is the value obtained from processing any given job J_{ux}.

The overall time needed to process a job J_{ux} is defined as:

$$CompTime(J_{ux}) = \sum_{i}^{q} E(J_{ux}, r_i) + Transfer(J_{ux}) \quad (1)$$

where $Transfer(J_{ux})$ is the sum of the time spent transferring data between resources ($r_i \in R$), where the first resource is located at the source of the data s and the last one is the destination d.

$$Transfer(J_{ux}) = \sum_{i}^{q} \sum_{k \neq i,k}^{q} T(r_i, r_k) \quad (2)$$

The overall cost of computing job J_{ux}, $Cost(J_{ux})$, is defined as:

$$Cost(J_{ux}) = CostExec + CostNet \quad (3)$$

where the computational cost ($CostExec$) is defined as:

$$CostExec = \sum_{i}^{q} [CE(r_i) * E(J_{ux}, r_i)] \quad (4)$$

$$CE(r_i) = BaseCost(r_i) * (1 + \frac{1}{Ratio_i}) \quad (5)$$

$$Ratio_i = \frac{Capacity_i}{\sum_{j=1}^{q} Capacity_j} \quad (6)$$

The cost of a resource r_i is defined by $CE(r_i)$ in Equation 5. This cost varies depending on the ratio between the capacity of r_i ($Capacity_i$) and the total capacity of the set R of resources ($\sum_i^q Capacity_i$). This ratio is represented by $Ratio_i$. The larger the capacity of a site, the lower the cost and the other way around.

The cost of transferring data associated with a job ($CostNet$) is defined as:

$$CostNet = \sum_{i}^{q} \sum_{k \neq i,k}^{q} [T(r_i, r_k) * BaseCostNet(r_i, r_k)] \quad (7)$$

subject to $E(r_k) \neq 0$.

In this work, our objective, from the infrastructure's perspective, is twofold. On the one hand, we want to maximize the throughput of our infrastructure, defined as maximizing the overall number of jobs processed by the system – as described in Equation 8. On the other hand, we want to maximize the overall *value* obtained from the processed data – as described in Equation 9.

$$\max \sum_{u}^{n} \sum_{x}^{m} \sum_{i}^{q} P(r_i) * E(J_{ux}, r_i) \quad (8)$$

$$\max \sum_{u}^{n} \sum_{x}^{m} Value(J_{ux}) \quad (9)$$

where the overall objective considers all cameras ($n \in C$) and all jobs $m \in J_u$ generated by each camera c_u are inserted into the system. These objectives are subject to ensuring the QoS requirements of each processed job, which is detailed in Section 5, Equations 13 and 14.

It is important to clarify that we do not associate particular operations (job executions) with particular resources in our system, i.e. it is not necessary for all collected data to be pre-processed at edge devices prior to their transmission to network or the data center resources. Data pre-processing, for instance, could be carried out on any resource depending on their capability, capacity and cost. The proposed architecture would be most effective if *similar* types of operations could be executed across all the devices in the system (with varying QoS profiles, depending on device type). In the limiting case, it may become necessary to carry out all such analysis at the data center, with edge and network devices primarily

enabling data capture and transmission. However, the aim of the optimization process is to push some of these operations to the edge of the network, whilst not violating some of the other constraints associated with application execution deadline or cost.

5 Scheduling Optimization Strategy

To achieve the previously described objectives, we add an additional stage to the workflow described in Figure 1. This stage is used to estimate the expected *value* of a video chunk, which represents the likelihood of finding relevant information within this chunk. This value is used to perform a systematic sampling that reduces the size of data without affecting its content. In practice, the expected value can be estimated using historical information combined with the current status of the recorded area.

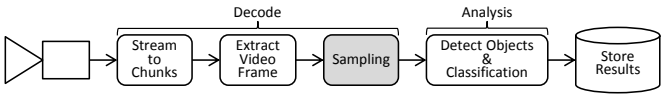


Fig. 4: Customized High Level Stream Processing Workflow

The semantics of *value* can change for different uses of video analysis. We consider two main aspects that influence this parameter: (i) static: these include characteristics such as the importance of a particular video source (e.g. position/geo-location of a particular camera), the video sequence captured during a particular event within a particular time window (e.g. a football event), etc. These characteristics are therefore known apriori, i.e. before the analysis is carried out; (ii) dynamic: these include characteristics that are derived after a part of the video sequence has been analysed – e.g. detection or classification associated with particular types of objects, with such object(s) not being known before analysis commences. Such dynamic aspects generally require an interactive analysis of a video sequence. In this work, we primarily focus on (i), although the workflow we propose could also be extended to (ii), but would require human/operator assessment during the workflow.

Figure 4 shows the workflow considered in this work, with the extra stage to assess *value*. We define high value (HV) and low value (LV) video chunks as follows:

$$Value(J_{ux}) \begin{cases} [0.5, 1] \rightarrow \text{High Value (HV)} \\ [0, 0.5) \rightarrow \text{Low Value (LV)} \end{cases} \quad (10)$$

The *value* of a video chunk, $Value(J_{ux})$, is used to decide how much data of this chunk we keep. Specifically, we perform a systematic sampling that reduces the size of the data. This sampling interval k is calculated by $\frac{N}{y}$, where N is the total number of frames and y is the sample size. The sample size y is calculated as follows:

$$Value(J_{ux}) \begin{cases} [0.5, 1] \rightarrow & y = \lfloor Value(J_{ux}) * N \rfloor \\ [0, 0.5) \rightarrow & y = \lfloor 0.5 * N \rfloor \end{cases} \quad (11)$$

Moreover, the value of a video chunk is also used to decide how a job should be scheduled. Currently, we consider two strategies: (i) minimizing the computational time required to process a job; and (ii) minimizing the cost of computing the job.

$$Value(J_{ux}) \begin{cases} [0.5, 1] \rightarrow (i)\ min\,(CompTime(J_{ux})) \\ [0, 0.5) \rightarrow (ii)\ min\,(Cost(J_{ux})) \end{cases} \quad (12)$$

both scheduling strategies are subject to performing computation within the given deadline (13), and keeping costs within the given budget (14).

$$CompTime(J_{ux}) \leq Deadline(J_{ux}) \quad (13)$$

$$Cost(J_{ux}) \leq Budget(J_{ux}) \quad (14)$$

In fact, our admission control strategy enforces Equations 13 and 14 by only accepting those jobs that can be completed while satisfying these constraints.

6 Configuration of Testbed

We configured a testbed using our previously proposed multi-layer computational model. In this model, we have three different kinds of computing resources: (i) Edge resources, close to data source; (ii) In-transit resources, close to data in movement; and (iii) Core data centers or sites, located deep into the infrastructure and far from data sources. We have used AWS EC2 cloud platform to emulate an actual scenario where resources are virtual machines and the network is controlled using Mininet [29] and Linux traffic control. Specifically, we used a total of 11 VM instances that emulated different geographically distributed sites, as described in Figure 5. Two VMs represented camera aggregators, located at the edge of the infrastructure, named Source1 and Source2. Another VM represented the datacenter where results are ultimately stored, named destination. The other eight VMs were in-transit resources: Mid1 through Mid8 – located between sources and destination, see Figure 5. Specifically, four in-transit resources (Mid1, Mid2, Mid5 and Mid6) were located along the path from Source1 to the Destination, and the other four (Mid3, Mid4, Mid7 and Mid8) were located along the path from Source2 to the Destination.

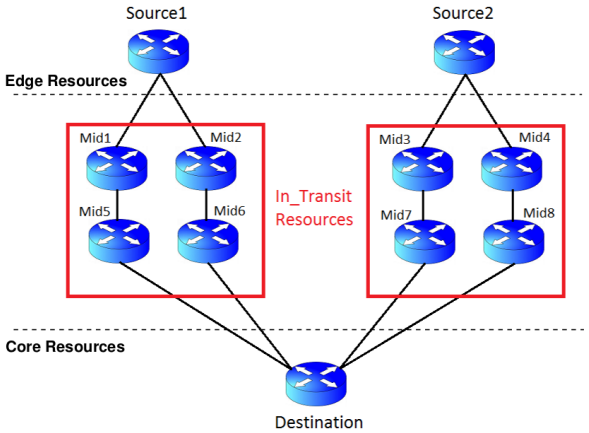


Fig. 5: Infrastructure. Solid lines indicate network links of 20MB of bandwidth. We assume each SDN router is co-located with computational resources.

Our deployment emulated an actual geographically distributed infrastructure by configuring network bandwidths

connecting the sites and the performance of the computational resources according to experimentally obtained information (between Rutgers and AWS East region). Table 1 summarizes the characteristics of the resources at each level of our infrastructure. We considered each worker had the performance of an Amazon EC2 c4.xlarge instance, with a base price of $0.21. The hardware characteristics of this instance match the one used to characterize the workload [2]. The price was calculated using the cost model defined in Equation 5. Equation 5 assumes that computational resources are limited and adjust the price based on the total capacity of each type of resource, the fewer resources, the more expensive is to reserve them and vice versa.

TABLE 1: Computational Resource Properties.

Resource	# Workers per Site	# Sites	Price ($/Hour)
Edge	1	2	4.83
In-transit	2	8	2.52
Core site	4	1	1.365

All instances were deployed with Mininet [29] and the network between them was configured to emulate a SDN environment among these 11 mininet instances. Each VM had one mininet host and one mininet switch. Switches were connected to each other using Generic Routing Encapsulation (GRE) tunneling[28]. Bandwidth allocation for data links was implemented in the hosts using a token bucket filter. Routing tables and connections were controlled by a POX SDN controller (POX is a python based SDN controller). We had an additional VM designated as the controller of the network. The controller managed network connections using two types of connections: (i) UDP was used for gathering information; and (ii) TCP was used for regular communication and establishing data paths. TCP rules for each switch were installed in a proactive manner. That is, every time a switch connected to the controller (i.e. when switch starts), the controller would install rules (as described below).

We implemented our in-transit optimization approach, described in Section 5, as follows. We defined our protocol for communication between the controller and hosts using UDP packets. We established that switches would forward all UDP packets to the controller unless a specific destination was included in the packets. We used this rule to enable communication between client and controller. Specifically, when the source site (i.e. client) wanted to communicate with the controller, it would send a UDP packet without destination field instantiated. This packet would be automatically forwarded to the Controller. Upon receipt of this packet, the controller would send UDP packets to all in-transit resources (hosts) asking for their status and capabilities information. Since those UDP packets would have a specific destination, switches knew where to send them (i.e. in-transit resources). Then, the controller would gather all replies, analyze their information, and create a plan. This plan was returned to the client and included a data transfer path as well as information about the allocated in-transit computation.

For example, in the case of the video processing environment, each camera aggregator collects video from the cameras and decides where the workload is computed according to the established policies and resource availability. Using the network configuration defined above (in this section), camera aggregators can transparently contact SDN controllers (without knowing their location or address) to obtain a view of the infrastructure and take operational decisions. This is achieved by simply sending a UDP message without destination. Using this approach, the use of SDN does not involve complex changes in the client and data producers.

7 Evaluation

In this work, we considered that each camera aggregator (Source) had three cameras capturing and sending video to them. Specifically, at each camera aggregator we had one camera capturing video with QCIF quality, another with CIF quality, and another one with 4CIF quality. We considered that the video feeds were sent in chunks of 48 seconds, hence each camera generated a new video processing job every 48 seconds. A second of video had 25 frames, where all of these video frames were independent of each other, from an object detection perspective. Table 2 summarizes the application characteristics in terms of execution and data size for different quality of video – the application was characterized by Anjum et al. in [2]. We used the execution time and data size to assign a deadline to each type of video to ensure timely delivery of results. We performed two sets of experiments using different deadlines to observe the behavior of the system and to understand the capacity of the infrastructure when processing our use case application. Additionally, we validated our model against results obtained from experiments and analytically studied the effect of changing various parameters in the system. In order to evaluate the model analytically, we implemented the mathematical model and the optimization strategy proposed in Sections 4 and 5 into a custom made Python simulation.

TABLE 2: Video Stream Analysis Time and Characteristics obtained from [2]

Format	Decode	Analysis	Size
QCIF	0.4	4 s	80 MB
CIF	2 s	12 s	320 MB
4CIF	4 s	40 s	1200 MB

Each experiment lasted around 30 minutes. During this time, each camera sent a total of 39 video processing jobs. As each camera was generating data with different encoding formats, the computational jobs were also heterogeneous in both computational and data transfer needs. Once a job was generated, the camera aggregator had to decide where to execute the job, if possible. The camera aggregator also estimated the expected *value* of each job. This value can be estimated using historical information or current status of the recorded areas. In our experiments, we used a random distribution to assign an expected *value* to each job. Figure 6 shows the value distribution for each type of job.

We evaluated three different scenarios:

Cloud (C): This scenario considered a traditional approach where all data was transfered to a large central data center for processing. Thus, every time a job was generated the camera aggregator asked the data center whether it was possible to complete the job to meet the pre-established SLA guarantees. If the job was accepted, then its data was transferred to

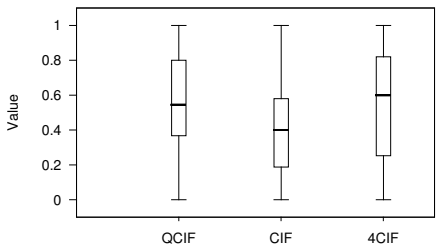


Fig. 6: Value Distribution for Each Job Type.

the data center for processing. The scheduling policy of this scenario was to minimize cost while meeting the deadline.

In-Transit plus Cloud (I+C): In this scenario we added a layer of computational resources to help the central data center process the workload. In this case, the camera aggregator asked the SDN controller to optimize the route from source to destination and determine the most efficient job execution (e.g., part of the job can be processed at one or more in-transit sites and part at the destination data center). The scheduling policy of this scenario was to minimize the cost of low value jobs by sending them to the core cloud data center, and to minimize the completion time of high value jobs by using in-transit resource(s) whenever possible. As before, jobs that could not be completed within the deadline were rejected.

Edge plus In-Transit plus Cloud (E+I+C): In this scenario, we added one more layer of processing to enable computation at the edge of the infrastructure (i.e. the camera aggregators). Specifically, we implemented the optimization strategy proposed in Section 5 to perform (some limited) computation at the edge. This computation involved performing a systematic sampling to reduce the size of the job (i.e. dropping frames). Sampling is a popular approximate computing technique for image processing, and it has shown to be very effective in accelerating computation while keeping the error of the solution within acceptable margins [15]. Next, jobs were scheduled across in-transit and cloud resources minimizing the cost for low value jobs and minimizing the completion time for high value jobs.

Due to capacity constraints, we considered that edge devices were not able to queue jobs and therefore had to push jobs to the next hop resource (an in-transit resource). On the other hand, we considered that cloud and in-transit resources had more capacity and therefore were able to queue jobs for processing.

The remaining subsections describe our experimental results for each of these scenarios.

7.1 Experiment 1 – Deadline based on completion time

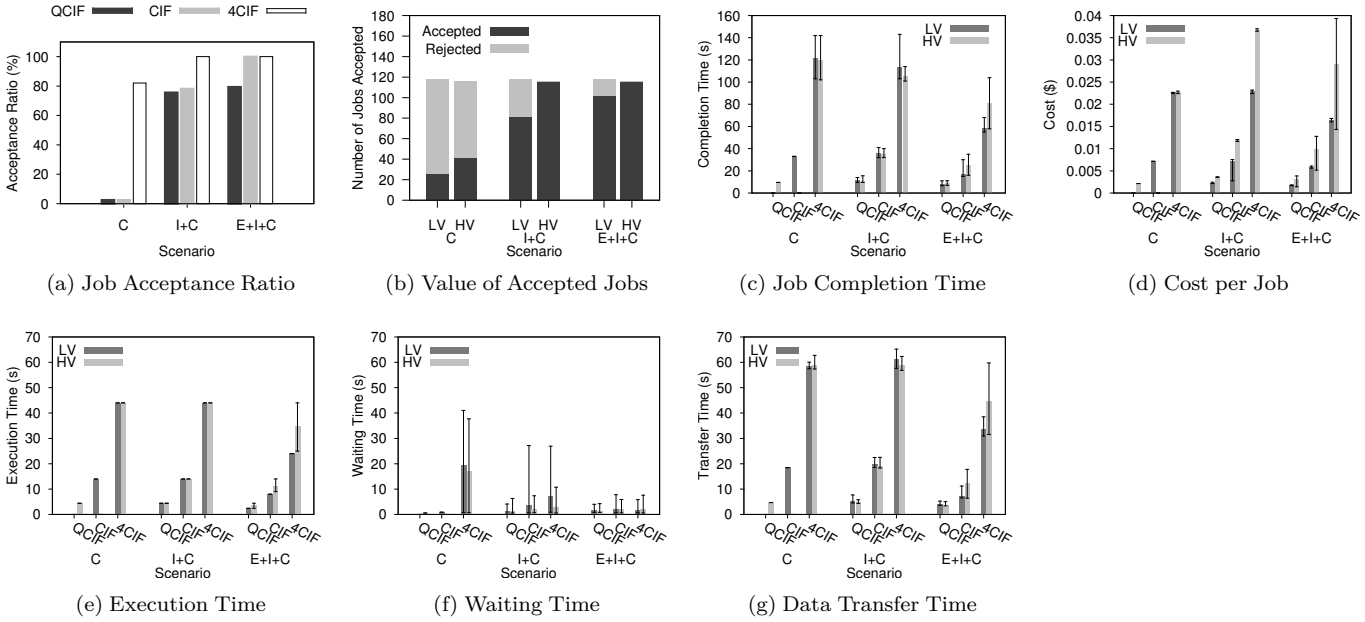
In the first set of experiments we used a deadline for each type of job that is 50% higher than its minimum completion time (execution plus data transfer) in the *cloud* scenario. Thus, the deadline for QCIF is 12 seconds, for CIF is 45 seconds, and for 4CIF is 156 seconds. These experiments were executed in a deployment of our CometCloud framework in Amazon EC2, as described in Section 6. Figure 7 illustrates the results.

Figure 7a compares the job acceptance ratio, represented as the percentage of jobs accepted for processing compared to the total number of jobs submitted. We can observe that the traditional approach of sending all data to a central data center located at the core of the infrastructure (labeled as C) was only able to process a small number of the required jobs. However, by adding an additional layer of processing to use resources located along the data path from source to destination (in-transit resources), the infrastructure was able to significantly increase the number of accepted jobs – especially smaller jobs (results labeled as I+C). This was not only due to the additional computational resources, but also to the fact that data was being processed earlier and its size was reduced. This contributes to reducing large waiting times, which allowed the acceptance of small jobs (QCIF and CIF) that in the first scenario had to be rejected due to potential violation of their deadlines. The last experiment introduced edge resources (E+I+C) and the possibility of doing some computation in-situ (where data was being generated). In this case, we observe that the infrastructure was able to further increase the number of completed jobs – only rejecting around 20% of the QCIF jobs.

As part of our optimization strategy, we wanted to increase the number of high value jobs processed by the infrastructure, as they were expected to provide us with more relevant information. Thus, Figure 7b compares results showing the *value* associated with accepted and rejected jobs. We observe that by simply using a different scheduling policy depending on the *value* parameter, we were able to prioritize high *value* jobs. In the I+C (In-transit plus Cloud), we accepted all high value jobs and only rejected 30% of the low value jobs. Additionally, in the E+I+C (Edge plus In-transit plus Cloud) we were able to accept all high value jobs, rejecting less than 17% of the low value jobs. In general, we can conclude that adding additional layers of computation closer to the data, improves the performance of the infrastructure and minimizes the network bottlenecks.

Figure 7c compares the completion time of all jobs in the system, calculated as the time since a job was inserted until it was processed. Figure 7c show how different scenarios influence the average completion time of jobs. It is worth noting the impact of our scheduling approach, depending on the *value* of the data, in I+C scenario. In this scenario we did not have any filtering of the video frames. However, the value of a job was used to decide the way such a job was scheduled, which in practice prioritized high value jobs. As a consequence, the average completion time of high value jobs was up to a 7% lower than the average completion time of low value jobs. Alternatively, in the E+I+C, we observe that the average completion time of high value jobs was 60% higher than low value jobs. The main reason was that during the sampling phase, high value jobs had twice as much data as lower value jobs and therefore the execution took longer. However, we observe that the completion time was significantly reduced when compared with the C and I+C scenarios (up to a 50%). In general, we observe that by adding edge and in-transit resources, we were able to reduce the completion time of all jobs compared with the approach of using only cloud resources (the C scenario) at the core of the infrastructure.

The completion time of a job was composed of three components, namely waiting time, execution time, and data transfer time. The waiting time or queue time, is defined as the time that a job spends waiting to be executed. Since our infrastructure is a multi-queue system, a single job may have to wait in more than one queue. Thus, the waiting time of a job



(a) Job Acceptance Ratio (b) Value of Accepted Jobs (c) Job Completion Time (d) Cost per Job

(e) Execution Time (f) Waiting Time (g) Data Transfer Time

Fig. 7: Summary of experimental results. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 10.

was calculated as the sum of its waiting time in every queue it was scheduled. Figure 7f collects the waiting time (queue time) of the jobs. As we can observe, in the I+C and E+I+C scenarios, the average waiting time of the high value jobs was up to an 77% lower than the waiting time of low value ones. In the C scenario (cloud), only the large jobs 4CIF, which had a large enough deadline, were able to wait in the queue, while small jobs (QCIF and CIF) were penalized and rejected due to having a very short deadline in comparison with the large jobs.

Next we analyzed the impact of the in-transit and edge resources on the execution time of the video processing jobs. Figure 7e demonstrates how our strategy of performing sampling and preprocessing of jobs at the edge helped to reduce the amount of execution needed for video processing jobs. Similarly, we observe in Figure 7g that the average amount of time spent transferring data between source and the place of computation was, in average, between 18% and 64% lower when using edge computation. As we can observe the network had a strong influence on the completion time of jobs, being at times larger than the execution time.

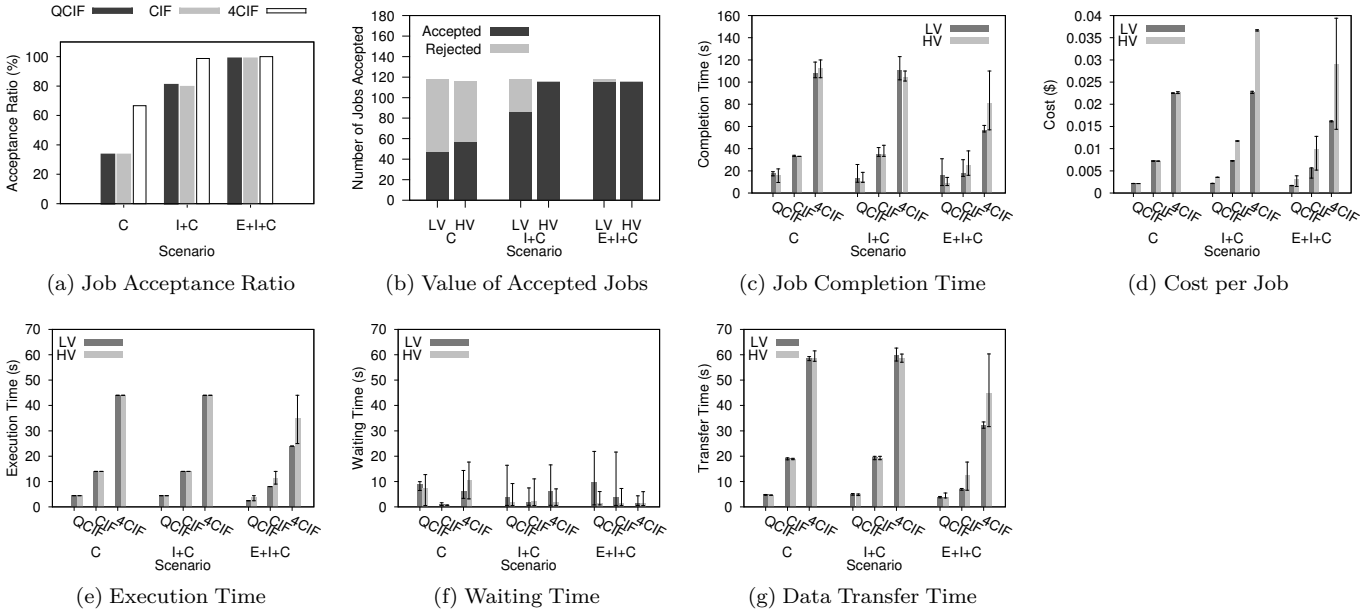
Finally, we also analyzed the impact of the scenarios on the total cost of the jobs. Table 1 collects the price of each resource per unit of time. In our model, the fewer resources a site have, the higher its price. In this way, performing computation at the edge is much more expensive that performing the same computation in a core cloud data center (composed by a large number of resources). Figure 7d collects the results. We observe that, in the I+C and E+I+C scenarios, low value jobs (scheduled aiming at minimizing cost) were typically computed at the destination (core cloud data center). However, high value jobs (scheduled aiming at minimizing completion time) chose to compute using edge and in-transit resources when they were available, which increased the price of computation. We also observed that in the E+I+C scenario, the average cost of the jobs was lower than in the I+C scenario. These savings were caused by the use of a sampling technique at the edge, which reduced the amount of data to be processed.

7.2 Experiment 2 – Deadline based on video size

In this experiment, we increased the deadline of the jobs of type QCIF and CIF to 48 seconds, which matches the size of the video to be processed, and in practice means near-real time processing of video feeds. Whereas in Experiment 1, we mapped deadline to minimum completion time, in this experiment we relate deadline to the size of the video to be processed. At the same time, we set the deadline of 4CIF job type to 120 seconds to reduce the significant waiting time observed earlier. Large waiting times for this type of jobs prevented us from accepting more small jobs (QCIF and CIF) as their deadline could not afford the wait. This experiment was executed in a deployment of our CometCloud framework in Amazon EC2, as described in Section 6 Figure 8 compares the results.

Figures 8a and 8b compares the results of admission control mechanism used in the infrastructure. We observe that by increasing the deadline of the smaller jobs, and specially reducing the deadline of the large jobs (4CIF), the Cloud scenario (labeled C) was able to increase the number of accepted jobs, resulting in a 45% acceptance ratio. On the other hand, the I+C scenario (In-transit plus Cloud) rejected 27% of low value jobs, while E+I+C scenario (Edge plus In-transit plus Cloud) only rejected 1.5% of the low value jobs. In both cases, all high value jobs were accepted.

Figure 8c compares the completion time for this experiment. We can observe that, despite increasing the deadline to the QCIF and CIF job types, their average completion time was similar to the one observed in Figure 7c. In fact, we observe in Figure 8f that the extra deadline was mainly used by the system to accommodate more jobs by increase waiting



(a) Job Acceptance Ratio (b) Value of Accepted Jobs (c) Job Completion Time (d) Cost per Job

(e) Execution Time (f) Waiting Time (g) Data Transfer Time

Fig. 8: Summary of experimental results. Deadlines are: QCIF = 48s, CIF = 48s, 4CIF = 120s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 10.

times. Moreover, we observe that the waiting time of the 4CIF jobs was reduced to enforce the new deadline. Therefore, we can conclude that the system is able to adaptively change job allocation to not only enforce the required QoS, but also to maximize the amount of jobs processed in the system.

Since the execution time and data transfer times were similar to those in the previous experiment, we observe in Figure 8d that the total cost for processing each type of job was also similar.

7.3 Experiment 3. Validation of the Model

In this Section, we validate our mathematical model by comparing the experimentally obtained results in Sections 7.1 and 7.2 against those obtained analytically using the model proposed in Section 4 and 5. This model has been implemented using a custom made Python simulator. Figure 9 collects the results of each experiment organized by columns.

Columns (a) and (b) of Figure 9 compare the results of the real experiment and the model for the experiment described in Section 7.1, respectively. Top and middle rows show the acceptance ratio per type of job and per value. We observe that there was no difference regarding to the number and type of jobs accepted. The bottom row shows the completion time of the jobs. In this case, we observe small differences between the model and the real experiment. The biggest differences were found in the smallest type of jobs (QCIF), where we had up to a 46% difference in the average completion time in the scenario including edge computing (E+I+C) – from 7.8 seconds in the real experiment to 4.2 seconds in the model. However, the other type of jobs showed less than 10% of difference. These differences, more noticeable in small jobs, were due to unaccounted overheads of the real infrastructure. The real experiment was executed in Amazon EC2, where the network performance was not guaranteed. Moreover, the orchestration and decision making operations could have also affected the completion times – e.g., interacting with the controller, deciding how to schedule the workload. These decision making overheads were hard to estimate to include in the model.

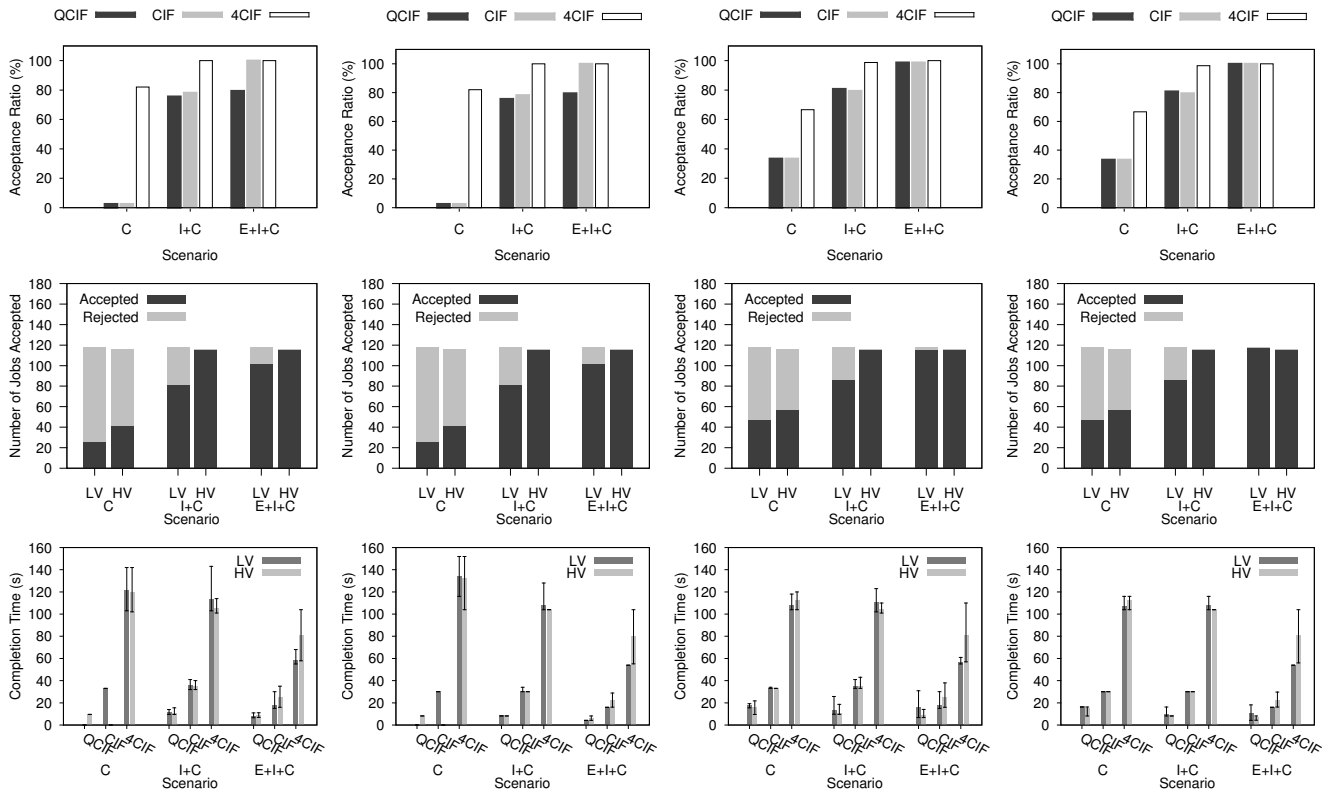
Columns (c) and (d) of Figure 9 compare the results of the real experiment and the model for the experiment described in Section 7.2, respectively. In this case, we observe in Figure 9c and 9d middle row a small difference in the number of accepted jobs. In particular in the real experiment two low value jobs were rejected in the E+I+C scenario, while the model considered that those jobs should have been accepted. Regarding the completion time, we observe also some differences in the type QCIF and scenarios named I+C and E+I+C. In this case the difference is up to a 33% for the QCIF jobs, and less than 10% difference for the rest of the cases.

We observed that the results obtained with the model were within a small error margin of the experimental results. Therefore, we consider to be proven that the model can reliably represent our experimental environment. Next, we continue the evaluation of our approach using our model, which allows us to easily introduce variations to the execution environment (i.e. infrastructure).

7.4 Experiment 4. Analytical Evaluation

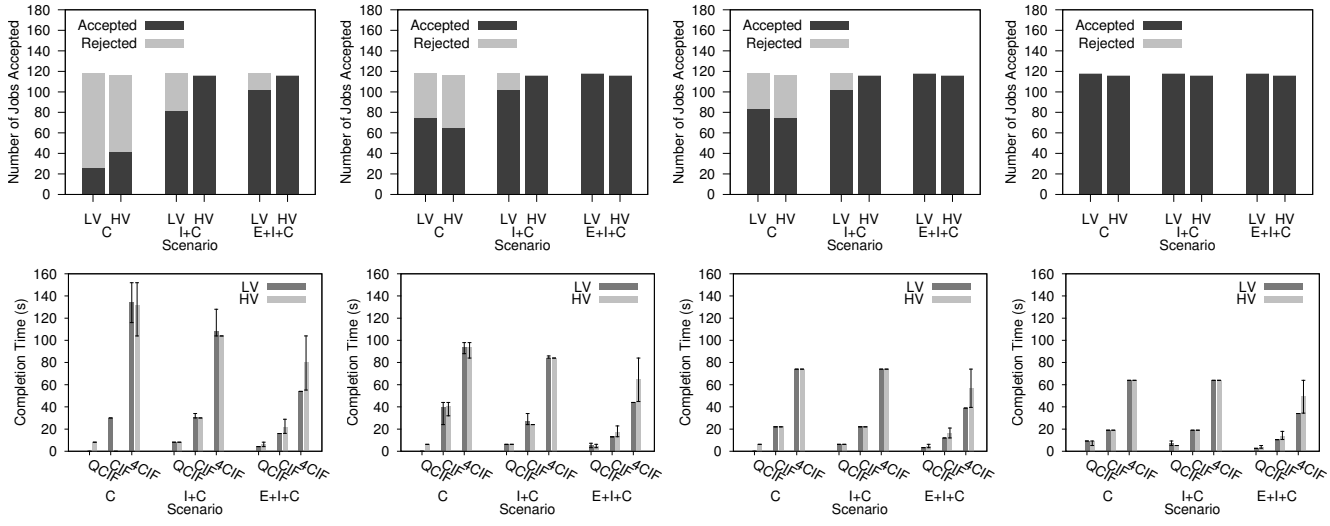
In this Section, we performed an analytical evaluation of our use case using our model. We changed different parameters of the infrastructure to observe their effect in the workload. We used the deadlines set in our first experiment (Section 7.1), as they had more room for improvement. The deadlines were QCIF = 12s, CIF = 42s, 4CIF = 150s. These experiments were analytically simulated using our mathematical model described in Section 4 and 5, and validated in Section 7.3.

Previously, we observed that the network had a strong influence on the completion time of our jobs. Hence, we first analyzed how the number of accepted jobs and their completion times were affected by changes in the network



(a) 12-42-150. Real (b) 12-42-150. Model (c) 48-48-120. Real (d) 48-48-120. Model

Fig. 9: Summary of model validation results. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 10. Each column represents a set of experiments, where Real means experimentally obtained and Model means analytically obtained.



(a) Baseline – Bandwidth 20MB (b) Bandwidth 30MB (c) Bandwidth 40MB (d) Bandwidth 60MB

Fig. 10: Summary of experimental results – Modifying Bandwidth. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 10.

bandwidth. Figure 10 collects the results – each column represents a different experiment. Column (a) of Figure 10 was our baseline scenario with a bandwidth of 20MB, which shows the results of the real experiments performed in Section 7.1. Columns (b), (c), and (d) of Figure 10 show results for an increasing bandwidth of network links, that is, 30MB, 40MB, and 60MB, respectively.

We can clearly observe how the bandwidth positively affected the acceptance rate and the completion time. We can observe in the bottom row of Figure 10, that in all cases the completion time of jobs was significantly reduced, between 20% to 50%, when increasing the network bandwidth. This

was the expected behaviour as our use case was highly data-intensive. We can also observe in the top row of Figure 10 that by reducing the completion time, the acceptance ratio increased. The scenario that most benefited from this increase was the Cloud deployment (labeled as C), as it is the one that required a larger amount of data to be transferred from source to the core of the infrastructure. Although we observed that from 20MB to 30MB we had a strong increase in the number of accepted jobs, between a 40% and 64%, this tendency slowed down in subsequent increases in bandwidth. It required increasing the network bandwidth by 200% to achieve 100% of job acceptance in the Cloud scenario. On the contrary, the other approaches also benefited from bandwidth increases and quickly reached 100% completion ratio for high value jobs. In the edge computing scenario (labeled as E+I+C), the system was able to accept all jobs, including low value ones, starting with 30MB network links. We can conclude that the proposed approach showed a *lower* dependency on the performance of the network (although the network capacity will influence the number of jobs submitted to the in-transit and cloud resources), primarily due to better use of the resources closer to the edge of the infrastructure, which allowed a lower use of the network links close to the core.

7.4.1 Number of Workers

In this Section we evaluated the effect that increasing the number of workers at different layers of the infrastructure has on the acceptance ratio of jobs. We also used different bandwidths to understand the relationship between the bandwidth and the number of workers. Table 3 collects the number of accepted jobs for each experiment. We use the first column named BW (bandwidth) to group experiments based on the network bandwidth used (i.e. 20, 30, and 40 MB). Within each network bandwidth group, we have experiments for the different scenarios considered in this paper, namely cloud (C), in-transit plus cloud (I+C), edge plus in-transit plus cloud (E+I+C). The four most right columns identify the number of workers used by each experiment. We have the Baseline column, which used the number of workers described in Table 1. Next we have the *2x C* column, which doubled the number of workers in the cloud site; the *2x I* column, which doubled the number of workers at each In-transit site; and the *2x C&I* column, which doubled the number of workers in both, the cloud and the in-transit sites.

Looking at the Table 3 from left to right, we can observe that changing the number of workers did not affect the acceptance ratio for any of the performed experiments. The main reason is that there was a small number of jobs waiting idle to be computed, as described in Figure 7f. Thus, the only changes observed in Table 3 were due to changes in the network link bandwidth – the effect of the bandwidth in the acceptance of jobs was studied in the previous section. Therefore, we can conclude that for data intensive applications, increasing the number of workers may not affect the number of accepted jobs.

7.4.2 Performance

Lastly, we studied the effect that the performance of the workers has on the acceptance ratio. We also used different bandwidths to make sure that the bandwidth did not impact the results. Table 4 collects the number of accepted jobs for each experiment. We use the first column named BW (bandwidth) to group experiments based on the network bandwidth used (i.e. 20, 30, and 40 MB). Within each network bandwidth group, we have experiments for the different scenarios considered in this paper, namely cloud (C), in-transit plus cloud (I+C), edge plus in-transit plus cloud (E+I+C). The four most right columns identify the number of workers used by each experiment. We have the Baseline column, where the performance of workers was as described in Table 1; in the *2x C* column, we doubled the performance of the workers located at the cloud site; in the *2x I* column, we doubled the performance of the workers located at each In-transit site; and in the *2x C&I* column, we doubled the performance of the workers located at both Cloud site and In-transit sites.

Looking at the Table 4 from left to right, we observe limited changes when modifying the performance of the workers, we marked those cases where the number of accepted jobs was different from the baseline. Table 4 shows that the performance of the workers had a limited effect in the number of accepted jobs for the Cloud scenario (labeled as C). For this scenario we observed that when the bandwidth was 20MB, the system was able to accept one additional low value (LV) job, although it rejected a high value (HV) job – in this scenario the scheduler did not differentiate between LV and HV jobs; for a bandwidth of 30MB, we observed an improvement of 8% for the HV jobs when increasing the performance; and for a bandwidth of 40MB, we observed no improvement at all. On the other hand, for the I+C (In-transit plus Cloud) and E+I+C (Edge plus In-transit plus Cloud), we observed no improvement in any of the cases. On the contrary, we

TABLE 3: Number of accepted jobs – Modifying Number of Workers. *2x C* means we doubled the number of workers in the cloud site; *2x I* means we doubled the number of workers in the in-transit sites; and *2x C&I* means we doubled the number of workers in both, the cloud and the in-transit sites. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 10.

			Modifying Number of Workers			
BW	Scenario	Value	Baseline	2x C	2x I	2x C&I
20 MB	C	LV	26	26	26	26
		HV	42	42	42	42
	I+C	LV	82	82	82	82
		HV	116	116	116	116
	E+I+C	LV	102	102	102	102
		HV	116	116	116	116
30 MB	C	LV	75	75	75	75
		HV	65	65	65	65
	I+C	LV	102	102	102	102
		HV	116	116	116	116
	E+I+C	LV	118	118	118	118
		HV	116	116	116	116
40 MB	C	LV	83	83	83	83
		HV	75	75	75	75
	I+C	LV	102	102	102	102
		HV	116	116	116	116
	E+I+C	LV	118	118	118	118
		HV	116	116	116	116

TABLE 4: Number of accepted jobs – Modifying Performance of Workers. *2x C* means we doubled the performance of workers in the cloud site; *2x I* means we doubled the performance of workers in the in-transit sites; and *2x C&I* means we doubled the performance of workers in both, the cloud and the in-transit sites. Deadlines are: QCIF = 12s, CIF = 42s, 4CIF = 150s. HV and LV correspond to high value and low value videos processing jobs, respectively – as defined in Equation 10.

			Modifying Performance of Workers			
BW	Scenario	Value	Baseline	2x C	2x I	2x C&I
20 MB	C	LV	26	**27**	26	**27**
		HV	42	**41**	42	**41**
	I+C	LV	82	**49**	82	82
		HV	116	116	116	116
	E+I+C	LV	102	**91**	102	102
		HV	116	116	116	116
30 MB	C	LV	75	**71**	75	**71**
		HV	65	**71**	65	**71**
	I+C	LV	102	**82**	102	102
		HV	116	116	116	116
	E+I+C	LV	118	**106**	118	118
		HV	116	116	116	116
40 MB	C	LV	83	83	83	83
		HV	75	75	75	75
	I+C	LV	102	**83**	102	102
		HV	116	116	116	116
	E+I+C	LV	118	**103**	118	118
		HV	116	116	116	116

observed that when increasing the performance of the Cloud workers, columns *2x C* and *2x C&I*, the number of accepted low value (LV) jobs decreased. The reason for this was that the scheduler, that used to prioritize allocating high value jobs among In-Transit resources to minimize completion time, decided to move some of the high value workload towards the Cloud due to its increased performance. This decision affected low value jobs as our scheduling policy tried to minimize the cost of low value jobs by allocating them in the Cloud. In general, we can conclude that, when the bandwidth limits the workload we can accept, increasing the performance of the resources in the federation have a limited effect in the ability of the system for increasing the number of accepted jobs.

8 Related Works

Our model proposes to have an infrastructure that combines edge computing and SDN/NFV architectures. On the one hand, edge computing is focused on migrating data processing from the core of the infrastructure towards the logical extremes of the network – which enables analytics and knowledge generation to occur close to the source of the data. For example, a practical architecture has been proposed by the European Telecommunications Standards Institute (ETSI) [7]. On the other hand, SDN/NFV architectures enable unprecedented control over the network, and for general purpose computation to be supported on network components. A practical architecture that combines SDN and NFV can be found in the SELFNET European project [16]. Using this type of architecture, we are able to leverage the SDN and NFV technologies to incorporate general purpose computation within the network data centers. Hence, we propose to extend the network controller capabilities to incorporate knowledge about the type of computation each network data center can perform.

IoT infrastructures identify the following components [4], [25]: (a) sensors, actuators and embedded communication hardware (b) on demand storage and computing tools for data analytics and (c) visualization and interpretation tools which can be widely accessed on different platforms and which can be designed for different applications. Based on these components a number of IoT applications such as smart grid and smart metering [27], video analytics are developed to efficiently optimize workflows and increased performances. For example, efficient energy consumption can be achieved by continuously monitoring sensors located within a house and using this information to impose how electricity is consumed. This information at the city scale is used for maintaining the load balance within the grid ensuring high quality of service.

On the other hand, video based IoT [1] combines image processing with computer vision and networking frameworks to enable surveillance, the most widely used camera network applications, track targets, identify suspicious activities, detect left luggage and monitor unauthorized access. The challenge is to impose automatic behavior analysis and event detection (as part of sophisticated video analytics) and on the other hand reduce costs and improve time-of-response.

In the area of video analytics in the cloud, systems usually identify hundreds or even thousands number of cameras covering over wide areas. The video streams derived are captured and processed at the local processing server and are later transferred to a cloud based storage infrastructure for a wide scale analysis. Such application identifies a complex workflow where an increasing amount of computation is required to process and analyze the video streams, an application where high performance and scalable computational is necessary for obtaining high throughputs. Video stream processing in the clouds represents an application case that can evolve into a area of research where high speed computation at scale, precision and efficiency become mandatory. Related video content retrieval have been proposed over time using Hadoop [23], encoding/decoding [26], distribution of video streams and on load balancing of computing resources for on-demand video streaming systems using cloud computing platforms [27].

The DISCOVERY project [10], [11] aims to design, implement, demonstrate and promote a unified system in charge of turning a complex, extremely large-scale and widely distributed infrastructure into a collection of abstracted computing resources which is efficient, reliable, secure and friendly to operate and use. The project looks at revising the OpenStack solution leveraging P2P mechanisms to address the architecture complexity of such systems and the velocity of open-source initiatives. In the same field of study, there are many related studies that have implemented SDN oriented solutions in order to ease the communication between different networking domains or to optimize various performance parameters within a complex system. Nunes et al. [17] have described the concept of SDN and the various layers involved in such system. Others have described security challenges faced by SDN [21, 22]. The most recognized protocol to enable a server (SDN controller) to control the switches is

OpenFlow [14]. In relation to SDNs, the SWITCH project[30] addresses a number of existing industrial requirements for developing and executing time critical applications in Clouds. SWITCH provides an interactive environment for developing applications and controlling their execution, a real-time infrastructure planner for deploying applications in Clouds, and an autonomous system adaptation platform for monitoring and adapting system behavior.

In the field of active networking, communication patterns are used for addressing specific user requirements [24]. An active network refers to a specific capability to execute tasks within the network over active elements such as switches that have processing capability. Lefevre et al. [12] developed an active network architecture (A-Grid) to support QoS-related metrics for Grid data transport services in addition to other data transport services such as reliable multicast and dynamic service deployment. The architecture employs QoS management at intermediate active routers, and in principal, it is similar to the in-transit processing employed in our approach.

Another emerging research topic is the availability of network resource reservation systems such as ESNET's OSCARS [8] and UltraScience Net [20]. These types of systems can provide on-demand dedicated bandwidth channels to user applications. The main idea in resource reservation systems is that a virtual single-switch abstraction is added on top of networks facilitating both a bandwidth reservation system and SDN processing.

This paper primarily focuses on edge and in-transit video processing and shares commonalities with [3] where tasks are distributed across multiple data centers (with varying types of capability), which are hierarchically organized, has implications on how users' data and processing are managed to improve service quality as well as reduce costs. In our research, we look at combining edge processing (at data capture site) with analysis carried out while data is enroute from the capture site to a data center and different processing models, and unlike mentioned papers our target is not only considering networking resource but also dealing with other computational and storage resources through the use of an SDN controller.

9 Discussion

The results presented in this paper show the limitations that traditional approaches, consisting on transferring all data from its source (data capture site) to the core of the infrastructure, face when applied to data-intensive applications with time constraints (such as video surveillance, smart Grid, and other IoT scenarios). In our experiments, we show how the network links can quickly become a bottleneck that slow down workload processing. This can lead to a number of jobs being rejected, as processing times will not be able to meet required QoS constraints (e.g., deadline).

We observed how our approach was able to overcome the limitations of a traditional approach by leveraging computational resources at the edge of the infrastructure (camera aggregators) and within the network data centers, through the use of SDN technology. The results show that by using our in-network computational model and the proposed scheduling strategy, the system was able to accept up to a 70% more workload. In our approach we used edge computing to perform a systematic sampling of the data, which in practice reduced computational requirements without affecting the key information derived from its content. Other applications could use edge computing to filter out invalid or out-of-range parameters, or perform similar operations that can help in using limited resources more effectively and increase the obtained value of the data. Nevertheless, the results show that even in cases where edge computing was not possible, leveraging SDN and NFV technologies to perform in-transit computation within the network data centers had a significant impact on the size of workload processed. In particular, we observed up to a 60% improvement in the job acceptance ratio.

We also discuss additional scenarios using our mathematical model to modify different parameters of the infrastructure that could affect the size of workload that the system was able to process. These experiments confirmed that the network was the single main factor limiting the amount of workload that the system was able to compute given certain deadline constraints. We observed that increasing the number of workers had no effect on the number of jobs accepted and increasing the performance of the workers had a very limited effect, less than 10%. Horizontal and vertical scaling of machines within a data center is certainly significant for data intensive applications, however, distributed data-intensive applications can benefit significantly from geographically distributed in-transit (computational) resources, as proposed in this paper.

10 Conclusions

In this paper we proposed a new in-network computational model that leverages resources distributed across the network, including edge devices and network data centers. We described how integrating SDN capability into our federated infrastructure can enable the use of resources located at the network data centers to perform in-transit computation of data that is being transferred. Moreover, we also proposed a strategy that leverages edge devices to prioritize the workload processing depending on the estimated value of the data. In this way, we could increase the amount of data processed to increase the overall *value* of the obtained results. We used a video surveillance application as use case, and tested several scenarios to show the feasibility and benefits of our proposed computational model by making use of edge and in-transit data analysis.

Currently, we are working to extend our model to incorporate dynamic bandwidth allocation that can help us further improve the use of the infrastructure while obtaining the maximum value out of the data. Moreover, we are exploring how to define multiple types of operations that can be performed at different levels of the infrastructure, and create a portfolio of operations that allows the infrastructure to dynamically take just-in-time decisions to meet a user's QoS requirements.

Acknowledgements: This work is supported in part by NSF via grants numbers ACI 1339036, ACI 1441376. The research at Rutgers was conducted as part of the Rutgers Discovery Informatics Institute (RDI2).

References

[1] I. F. Akyildiz, T. Melodia, and K. R. Chowdhury. A survey on wireless multimedia sensor networks. *Comput. Netw.*, 51(4):921–960, Mar. 2007.

[2] A. Anjum, T. Abdullah, M. F. Tariq, Y. Baltaci, and N. Antonopoulos. Video stream analysis in clouds: An object detection and classification framework for high performance video analytics. *IEEE Transactions on Cloud Computing*, 1(1):1–14, 2016.

[3] L. F. Bittencourt, M. M. Lopes, I. Petri, and O. F. Rana. Towards virtual machine migration in fog computing. In *10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, to appear*, Krakov, Poland, June 2016.

[4] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic. Cloud computing and emerging {IT} platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[5] J. Diaz-Montes, M. AbdelBaky, M. Zou, and M. Parashar. Comet-cloud: Enabling software-defined federations for end-to-end application workflows. *IEEE Internet Computing*, 19(1):69–73, 2015.

[6] J. Diaz-Montes, Y. Xie, I. Rodero, et al. Federated computing for the masses - aggregating resources to tackle large-scale engineering problems. *CiSE Magazine*, 16(4):62–72, 2014.

[7] ETSI. Mobile-edge computing - introductory technical white paper. Technical report, ETSI, 2014.

[8] C. Guok, D. Robertson, M. Thompson, J. Lee, B. Tierney, and W. Johnston. Intra and interdomain circuit provisioning using the oscars reservation system. In *Intl. Conf. on Broadband Communications, Networks and Systems*, Oct 2006.

[9] S. Jain, A. Kumar, S. Mandal, et al. B4: Experience with a globally-deployed software defined wan. In *ACM SIGCOMM 2013*, SIGCOMM '13, pages 3–14, 2013.

[10] A. Lebre, J. Pastor, and T. D. consortium. In inria research report. In *The DISCOVERY Initiative - Overcoming Major Limitations of Traditional Server-Centric Clouds by Operating Massively Distributed IaaS Facilitiese*, France, Sept 2015.

[11] A. Lebre, A. Simonet, and A.-C. Orgerie. Ieee intl. workshop on cloud computing interclouds, multiclouds, federations, and interoperability. In *Deploying Distributed Cloud Infrastructures: Who and at What Cost?*, Berlin, Germany, June 2016.

[12] L. Lefevre, C.-d. Pham, P. Primet, B. Tourancheau, B. Gaidioz, J.-P. Gelas, and M. Maimour1. Active networking support for the grid. In I. Marshall, S. Nettles, and N. Wakamiya, editors, *Active Networks*, pages 16–33. 2001.

[13] Z. Li and M. Parashar. Comet: A scalable coordination space for decentralized distributed environments. In *Intl. Workshop on Hot Topics in Peer-to-Peer Systems*, 2005.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[15] S. Mittal. A survey of techniques for approximate computing. *ACM Comput. Surv.*, 48(4):62:1–62:33, Mar. 2016.

[16] P. Neves, R. Calé, M. R. Costa, et al. The selfnet approach for autonomic management in an nfv/sdn networking paradigm. *International Journal of Distributed Sensor Networks*, pages 1–17, 2016.

[17] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. 2014.

[18] OpenCloud. Nfaas - network function as a service, 2014.

[19] I. Petri, M. Zou, A. Zamani, J. Diaz-Montes, O. F. Rana, and M. Parashar. Integrating software defined networks within a cloud federation. In *CCGrid 2015*, 2015.

[20] N. Rao, W. Wing, S. Carter, and Q. Wu. Ultrascience net: network testbed for large-scale science applications. *Communications Magazine, IEEE*, 43(11):S12–S17, Nov 2005.

[21] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. Sdn security: A survey. In *IEEE SDN for Future Networks and Services*, 2013.

[22] S. Shin and G. Gu. Attacking software-defined networks: A first feasibility study. In *ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 165–166, 2013.

[23] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Proceedings of the 2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*, MSST '10, pages 1–10, Washington, DC, USA, 2010.

[24] D. L. Tennenhouse and D. J. Wetherall. Towards an active network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(5):81–94, Oct. 2007.

[25] M. Tory and T. Moller. Rethinking visualization: A high-level taxonomy. In *IEEE Symposium on Information Visualization*, INFOVIS '04, pages 151–158, Washington, DC, USA, 2004. IEEE Computer Society.

[26] Y. Wu, C. Wu, B. Li, X. Qiu, and F. C. M. Lau. Cloudmedia: When cloud on demand meets video on demand. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 268–277, June 2011.

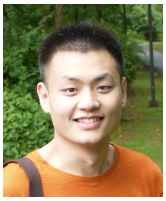
[27] M. Yun and B. Yuxin. Research on the architecture and key technology of internet of things (iot) applied on smart grid. In *Intl. Conf. Advances in Energy Engineering*, pages 69–72, June 2010.

[28] GRE Tunneling. http://lartc.org/howto/lartc.tunnel.gre.html, Last accessed on Nov. 2016.

[29] Mininet project. http://mininet.org, Last accessed on Nov. 2016.

[30] SWITCH project. http://www.switchproject.eu, Last accessed on Nov. 2016.

Ali Reza Zamani is currently a PhD student in Computer Science department at Rutgers University. Also he is a member of the Rutgers Discovery Informatics Institute (RDI2). He received his M.Sc. degree in Computer Science from Rutgers University. He received his B.Sc. from Sharif University of Technology, Iran. His research interests are in the areas of Software Defined Networking(SDN), Network Functions Virtualization(NFV) and apply them in order to improve the performance of cloud federation systems.



Mengsong Zou is currently a PhD student in Computer Science at Rutgers University, and a member of the Rutgers Discovery Informatics Institute (RDI2). He received both of his Bachelor and Master degrees in Computer Science from Huazhong University of Science and Technology, China. His current research interest lies in parallel and distributed computing, cloud computing and scientific workflow management.

Javier Diaz-Montes is Assistant Research Professor at Rutgers University and a member of the Rutgers Discovery Informatics Institute (RDI2). He received his PhD degree in Computer Science from the Universidad de Castilla-La Mancha, Spain ("Doctor Europeus", 2010). Before joining Rutgers, he was Postdoctoral Fellow at Indiana University. His research interests are in the area of parallel and distributed computing and include autonomic computing, cloud computing, virtualization, and scheduling.

Ioan Petri is a Research Associate in School of Computer Science & Informatics at Cardiff University. He holds a PhD in 'Cybernetics and Statistics' and has worked in industry, as a software developer at Cybercom Plenware. His research interests are cloud computing, peer-to-peer economics and information communication technologies.

Omer F. Rana is a Professor of Performance Engineering in School of Computer Science & Informatics at Cardiff University & a member of Cardiff University's "Data Innovation Institute". He holds a Ph.D. in "Neural Computing and Parallel Architectures" from Imperial College (University of London). His research interests include distributed systems and scalable data analysis.

Ashiq Anjum is a professor at the University of Derby. He completed his PhD from UWE Bristol. His research interests include big data, cloud computing and analytics.

Manish Parashar is Distinguished Professor of Computer Science at Rutgers University. He is also the founding Director of the Rutgers Discovery Informatics Institute (RDI2). His research interests are in the broad areas of Parallel and Distributed Computing and Computational and Data-Enabled Science and Engineering. Manish serves on the editorial boards and organizing committees of a large number of journals and international conferences and workshops, and has deployed several software systems that are widely used. He has also received a number of awards and is Fellow of AAAS, Fellow of IEEE/IEEE Computer Society and ACM Distinguished Scientist.